TASK:

Class -> Hotel

Methods -> bookRoom(String roomType){sout(room of \*\*\* type is booked)}

bookroom(String roomType, int days){sout(room of \*\*\* type has been booked for \*\*\* days)}

object creation -> accessing methods

package hotelExample;  
  
public class Hotel {  
 void bookRoom(String roomType){  
 System.*out*.println("Room of type "+roomType + " has been booked!" );  
 }  
  
 void bookRoom(String roomType, int days){  
 System.*out*.println("Room of type "+ roomType+ "has been booked for "+ days + " days !");  
 }  
}

package hotelExample;  
  
public class Main {  
 public static void main(String[] args) {  
 Hotel taj = new Hotel();  
 taj.bookRoom("Royal");  
 taj.bookRoom("Royal",10);  
 }  
}

Method Overriding:

A child class provides a specific implementation of a method already defined in its parent class.

package overridingEx;  
  
public class Parent {  
 void show(){  
 System.*out*.println("This is parent class method");  
 }  
}

package overridingEx;  
  
public class Child extends Parent{  
 @Override  
 void show(){  
 System.*out*.println("This is child class method");  
 }  
}

package overridingEx;  
  
public class Main {  
 public static void main(String[] args) {  
 Parent obj = new Child(); // Parent reference & child object  
 obj.show();  
 }  
}

o/p:

This is child class method

* Use of Super Keyword:
* When a subclass and superclass have methods with the same name, super can be used to call the superclass method.

package superKeyEx;  
  
public class Parent {  
 void display(){  
 System.*out*.println("Parent Class");  
 }  
}

package superKeyEx;  
  
public class Child extends Parent {  
 void display(){  
 System.*out*.println("Child Method");  
 }  
  
 void show(){  
 super.display(); // Calling the Parent class method  
 display(); // Calls the child class method  
 }  
}

package superKeyEx;  
  
public class Main {  
 public static void main(String[] args) {  
 Child obj = new Child();  
 obj.show();  
 }  
}

o/p:

Parent Class

Child Method

Explanation:

Super.display() call the display method of parent class, while display() without the super refers to the child class’s display method.

Task: Create a person class with details like name and age. Then create two subclasses: Student and Teacher. Both should inherit from Person and add specific attributes (e.g. grade for student and subject for teacher). Write a program to display the details of a student and a teacher, showing inheritance in action.

Constructor:

* A special type of method which is used to initialize an object when it is created.
* It has same name as the class name and does not have any return type (not even void)

Why?

* Whenever an object of a class is created using the new keyword, java automatically calls a constructor to initialize the object.
* Without constructor we have to manually set the values of instance variables after creation of an object

1. Default Constructor
2. Parameterized constructor

Solution of TASK:

package que;  
  
public class Person {  
 // attributes  
 String name;  
 int age;  
  
 //parameterized Constructor  
 Person(String name, int age){  
 this.name = name;  
 this.age = age;  
 }  
}

package que;  
  
public class Student extends Person{  
 //attribute  
 String grade;  
  
 //constructor  
 Student(String name, int age, String grade){  
 super(name, age); //Calling the parent class constructor  
 this.grade=grade;  
 }  
  
 void displayDetails(){  
 System.*out*.println("Student: "+ name + ", Age: "+ age +", Grade: "+ grade);  
 }  
}

package que;  
  
public class Teacher extends Person {  
 //Attribute  
 String subject;  
  
 Teacher(String name, int age, String subject){  
 super(name,age); // Calls the parent class constructor  
 this.subject=subject;  
 }  
  
 void displayDetails(){  
 System.*out*.println("Teacher: "+ name + ", age: "+ age +", subject: "+ subject);  
 }  
}

package que;  
  
public class Main {  
 public static void main(String[] args) {  
 Student krishna = new Student("Krishna",24,"A+");  
 Teacher gopal = new Teacher("Gopal", 35,"JAVA");  
  
 krishna.displayDetails();  
 gopal.displayDetails();  
 }  
}

Abstraction:

Hiding the implementation detail and showing only the essential features

Ex. TV remote(Only buttons are visible, not the internal circuits)

Encapsulation:

Wrapping of data(variables) and methods in a single unit(Class), restricting the direct access to data.

Example: medical Capsule (The contents are enclosed/ encapsulated in a capsule)

Real-life analogy:

Car interface:

1. Driver operates a car without knowing or understanding how the engine works(Abstraction)
2. The engine and other mechanical components are encapsulated within car’s body (Encapsulation)

Abstract Class:

A class which cannot be instantiated and may contain abstract methods(without body) and non-abstract method.

package abstractEx;  
  
//Abstract Class  
abstract class Vehicle {  
  
 //Abstract Method (Without implementation)  
 abstract void start();  
  
 //Non-abstract method(With Implementation)  
 void stop(){  
 System.*out*.println("Vehicle Stopped!!");  
 }  
}

package abstractEx;  
  
  
//Subclass car extends the abstract class vehicle  
public class Car extends Vehicle{  
 //Providing the implementation of abstract method  
 void start(){  
 System.*out*.println("Car Started");  
 }  
}

package abstractEx;  
  
public class Main {  
 public static void main(String[] args) {  
 //Creating an instance of a CAR class  
 Car car = new Car();  
  
 //Call the start method from car class  
 car.start();  
  
 //Call the stop method(Inherited from Vehicle class)  
 car.stop();  
 }  
}

* Interfaces

A completely abstract class with only abstract methods.

Task : create Animal interface:

Define abstract method sound()

Create a class Dog which will implement Animal

Provide the implementation of abstract method.